Assignment 2: Usermode interrupt handlers

February 13, 2018

1 Introduction

In this assignment, we allow applications to handle their own exception. The applications register their exception handler (`CS:EIP`) with the kernel module. The kernel module sets the appropriate bits in the `IDT` entry (e.g., `DPL`, `EIP`, `CS`) such that the user-mode handler will be called for subsequent exceptions.

`struct idt_entry` represents the different fields of an `IDT` entry. For this assignment, you may need to modify the sel (`CS` selector), dpl (for `CPL-3` handling), lower16 (lower 16-bits of target exception handler), and higher16 (higher 16-bits of target exception handler). The kernel module handles two types of ioctl. `REGISTER_HANDLER` registers an exception handler (`CS:EIP`). The module saves the previous exception handler (`CS:EIP`) and sets the user-mode exception handler (by modifying the `PC`, `CS`, and `DPL` fields of `idt_entry`).

One of the problems in overwriting `IDT` descriptors is the race conditions due to partially updated entry. E.g., suppose we have updated higher16 but not lower16 and an interrupt in between preempt the current module and schedule a new process. If the new process triggers an exception, then the partially updated entry may result in an inconsistent state. There are two ways to solve this problem:

- Disable interrupts during the period you are updating the `IDT`. On X86 platform, you can use `cli/sti` instructions to disable/enable interrupts. Alternatively, you can use `local_irq_save` and `local_irq_restore` macros, which additionally save/restore flags (to correctly handle the situations when the interrupts were already disabled).
- Create a new `IDT` (by copying entries from the old `IDT`), make modifications to new `IDT` and finally load the new `IDT` (using `lidt`; atomic) to tell the processor to use this `IDT`.

For this assignment, you have to disable/enable interrupts before/after updating IDT entries. `UNREGISTER_HANDLER` simply restores the original exception handler.

A potential problem in using user-mode exception handler is to need to modify the scheduler because user-mode exception handlers are private to a process. When the OS decides to schedule a different process, it has to either restore the default exception handler (in the kernel) or a different exception handler which is private to the application. Using `REGISTER_HANDLER`, an application can register a divide by zero exception handler with the kernel. For this assignment, we assume that none of processes are going to do divide by zero during the interval our application has registered its custom handler (and hence no need to modify the scheduler).

2 Turn in

The user folder contains exception.c. It registers (`__ex_handler`) routine with the kernel module using `REGISTER_HANDLER` ioctl. At this point, a divide by zero on that processor causes the invocation of `__ex_handler` routine. Notice, since the exception handler is handled in ring-3 itself (no ring transitions), the hardware will push, old `EFLAGS`, old `CS` and old `EIP` (lecture-2) on the application stack. For this assignment, you need to skip the instruction which is causing the divide by zero exception and resume the execution. In other words, the exception handler will just ignore c = 1/0 and resumes the execution of subsequent instructions. On X64 hardware, the compiler emits `idiv` instruction for division. The excepting EIP pushed on the stack by hardware is the `idiv` instruction. To skip this instruction, you need to simply increment the program counter on the stack with the length of `idiv` instruction and return from the exception. Notice that you can not use `iret` in user-mode, so you need to emulate that with branch instructions like (`ret`, `jmp`, etc.). Additionally, you need to restore the EFLAGS from the stack which is pushed by the hardware (see `popf` instruction). If your exception handler (`__ex_handler`) implementation does not modify flags then you do not need to restore the flags but for this assignment you must have to restore the eflags. To know the length of `idiv` instruction, disassemble exception executable using ``` ``objdump -dx exception'' ``` and look for `idiv`.

To summarize:

- Implement `REGISTER_HANDLER`.
- Implement `UNREGISTER_HANDLER`.
- Must disable/enable interrupts before/after updating `IDT`.
- Do not change scheduler.
- Implement `__ex_handler` in handler.S.
- Must restore eflags from the stack (pushed by hardware on exception).

2.1 How to submit.

Get a copy of the assignment repository from `https://github.com/Systems-IIITD/aos02`. Use the VM provided for assignment 1. Before implementing anything first make a local branch (`git checkout -b [your_branch_name]`). For submission send a diff to the master branch. Run ``` ``git diff master > submit.txt'' ``` to redirect your changes to the submit.txt file. Submit the submit.txt file (please do not submit the entire folder).